Lecture 14: Nested Lists, Tuples, and Dictionaries
(Sections 11.1-11.5, 12.1-12)

CS 1110
Introduction to Computing Using Python

[E. Andersen, A. Bracy, D. Gries, L. Lee, S. Marschner, C. Van Loan, W. White]
Announcements

• A3 Tentative release date: Mon Mar 19-Thu Mar 22; tentative time for completion: somewhere between 1 and 2 weeks. Similar to A3 from Spring 2017.

• Prelim 1 Grading this weekend. Grades will come out before the drop deadline.
Next week: Recursion

- Tuesday and Thursday: Recursion.
- Reading: 5.8-5.10
Nested Lists

- Lists can hold any objects
- Lists are objects
- Therefore lists can hold other lists!

\[
\begin{align*}
  b &= [3, 1] \\
  c &= [1, 4, b] \\
  a &= [2, 1] \\
  x &= [1, a, c, 5] \\
  x &= [1, [2, 1], [1, 4, [3, 1]], 5]
\end{align*}
\]
Two Dimensional Lists

Table of Data

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td>4</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>8</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>0</td>
</tr>
</tbody>
</table>

Images

Store them as lists of lists (row-major order)

\[ d = \begin{bmatrix} [5,4,7,3], [4,8,9,7], [5,1,2,3], [4,1,2,9], [6,7,8,0] \end{bmatrix} \]

Each row, col has a value

Each row, col has an RGB value
Overview of Two-Dimensional Lists

- Access value at row 3, col 2:
  \[ d[3][2] \]
- Assign value at row 3, col 2:
  \[ d[3][2] = 8 \]
- Number of rows of \( d \):
  - \( \text{len}(d) \)
- Number of cols in row \( r \) of \( d \):
  - \( \text{len}(d[r]) \)
How Multidimensional Lists are Stored

- \( b = \begin{bmatrix} [9, 6, 4], [5, 7, 7] \end{bmatrix} \)

- \( b \) holds id of a one-dimensional list
  - Has \( \text{len}(b) \) elements

- \( b[i] \) holds id of a one-dimensional list
  - Has \( \text{len}(b[i]) \) elements
Ragged Lists: Rows w/ Different Length

- \( b = [[17, 13, 19], [28, 95]] \)
Slices and Multidimensional Lists

- Only “top-level” list is copied.
- Contents of the list are not altered
- \( b = [[9, 6], [4, 5], [7, 7]] \)

\[ x = b[:2] \]
Create a nested list

```python
>>> b = [[9,6],[4,5],[7,7]]
```

Get a slice

```python
>>> x = b[:2]
```

Append to a row of `x`

```python
>>> x[1].append(10)
```

What is now in `x`?

A: `[[9,6,10]]`
B: `[[9,6],[4,5,10]]`
C: `[[9,6],[4,5,10],[7,7]]`
D: `[[9,6],[4,10],[7,7]]`
E: I don’t know
Slices & Multidimensional Lists (A1)

• Create a nested list
  >>> b = [[9,6],[4,5],[7,7]]

• Get a slice
  >>> x = b[:2]

• Append to a row of x
  >>> x[1].append(10)

What is now in x?

A: [[9,6,10]]
B: [[9,6],[4,5,10]]
C: [[9,6],[4,5,10],[7,7]]
D: [[9,6],[4,10],[7,7]]
E: I don’t know
Slices & Multidimensional Lists (Q2)

- Create a nested list
  ```python
  >>> b = [[9,6],[4,5],[7,7]]
  ```
- Get a slice
  ```python
  >>> x = b[:2]
  ```
- Append to a row of x
  ```python
  >>> x[1].append(10)
  ```
- x now has nested list
  ```python
  [[9, 6], [4, 5, 10]]
  ```

- What is now in b?

  - A: `[[9,6],[4,5],[7,7]]`
  - B: `[[9,6],[4,5,10]]`
  - C: `[[9,6],[4,5,10],[7,7]]`
  - D: `[[9,6],[4,10],[7,7]]`
  - E: I don’t know
Slices & Multidimensional Lists (A2)

• Create a nested list
  >>> b = [[9,6],[4,5],[7,7]]

• Get a slice
  >>> x = b[:2]

• Append to a row of x
  >>> x[1].append(10)

• x now has nested list
  [[9, 6], [4, 5, 10]]

• What is now in b?

A: [[9,6],[4,5],[7,7]]
B: [[9,6],[4,5,10]]
C: [[9,6],[4,5,10],[7,7]]
D: [[9,6],[4,10],[7,7]]
E: I don’t know
Data Wrangling: Transpose Idea

4 lists: 2 elements in each
2 lists: 4 elements in each

How to transpose?

- 1\textsuperscript{st} element of each list gets appended to 1\textsuperscript{st} list
- 2\textsuperscript{nd} element of each list gets appended to 2\textsuperscript{nd} list
def transpose(orig_table):
    """Returns: copy of table with rows and columns swapped
    Precondition: table is a (non-ragged) 2d List""
    numrows = len(orig_table)
    numcols = len(orig_table[0]) # All rows have same no. cols
    new_table = [] # Result accumulator
    for m in list(range(numcols)):
        row = [] # Single row accumulator
        for n in list(range(numrows)):
            row.append(old_table[n][m]) # Build up new row
        new_table.append(row) # Add new row to new table
    return new_table
Tuples fall between strings and lists

- write them with just commas: 42, 4.0, ‘x’
- often enclosed in parentheses: (42, 4.0, ‘x’)
Returning multiple values

• Can use lists/tuples to return multiple values

INCHES_PER_FOOT = 12

def to_feet_and_inches(height_in_inches):
    feet = height_in_inches // INCHES_PER_FOOT
    inches = height_in_inches % INCHES_PER_FOOT
    return (feet, inches)

all_inches = 68
(ft, ins) = to_feet_and_inches(all_inches)
print(You are "+str(ft)+" feet, "+str(ins)+" inches.")
# Dictionaries (Type `dict`)

## Description

- List of **key-value** pairs
  - Keys are unique
  - Values need not be
- Example: net-ids
  - net-ids are **unique** (a key)
  - names need not be (values)
  - js1 is John Smith (class ’13)
  - js2 is John Smith (class ’16)

## Python Syntax

- Create with format: `{k1:v1, k2:v2, ...}`
- Keys must be **immutable**
  - ints, floats, bools, strings
  - **Not** lists or custom objects
- Values can be anything
- Example:
  ```python
d = {'ec1':'Ezra Cornell',
       'ec2':'Ezra Cornell',
       'ela63':'Erik Andersen'}
  ```
Using Dictionaries (Type `dict`)

- Access elements like a list
  
  \[ d = \{ 'ecl': 'Ezra', 'ec2': 'Ezra', 'ela63': 'Erik' \} \]

  - \( d['ecl'] \) evaluates to \( 'Ezra' \)
  - But cannot slice ranges!

![Diagram of dictionary structure]
Using Dictionaries (Type `dict`)

- Dictionaries are **mutable**
  - Can reassign values
  - `d['ec1'] = 'Ellis'`

```python
d = {'ec1':'Ezra', 'ec2':'Ezra', 'ela63':'Erik'}
d['ec1'] = 'Ellis'
```

![Diagram of dictionary structure]
Dictionaries are **mutable**

- Can reassign values
- `d['ec1'] = 'Ellis'`

```python
id8
dict

{'ec1':'Ezra','ec2':'Ezra','ela63':'Erik'}
```

```
id8
dict

{'ec1': 'Ezra', 'Ellis'}

'ec2': 'Ezra'

'ec2': 'Ezra'

'ela63': 'Erik'
```
Using Dictionaries (Type `dict`)

- Dictionaries are **mutable**
  - Can reassign values
    - `d['ec1'] = 'Ellis'`
  - Can add new keys
    - `d['aal'] = 'Allen'`

```python
id8
d = {'ec1': 'Ezra', 'ec2': 'Ezra', 'ela63': 'Erik'}
```
Using Dictionaries (Type `dict`)

- Dictionaries are **mutable**
  - Can reassign values
  - `d['ec1'] = 'Ellis'`
  - Can add new keys
  - `d['aal'] = 'Allen'`

```python
d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik','aal1':'Allen'}
```
• Dictionaries are **mutable**
  - Can reassign values
    - `d['ec1'] = 'Ellis'`
  - Can add new keys
    - `d['aa1'] = 'Allen'`
  - Can delete keys
    - `del d['ela63']`

```python
import uuid

d = {'ec1':'Ezra','ec2':'Ezra',
     'ela63':'Erik','aa1':'Allen'}
```
Using Dictionaries (Type \texttt{dict})

- Dictionaries are \textbf{mutable}
  - Can reassign values
  - \( d['ec1'] = 'Ellis' \)
  - Can add new keys
  - \( d['aal'] = 'Allen' \)
  - Can delete keys
  - \texttt{del} \( d['ela63'] \)

\[
d = \{ 'ec1':'Ezra', 'ec2':'Ezra', 'aal':'Allen' \}
\]